CS 240 Programming in C

Introduction to Pointers

October 21, 2019
& operator

- A pointer is a variable that contains the address of a variable.
- The unary operator & gives the address of an object.
- The & operator only applies to objects in memory: variables and array elements.
- It cannot be applied to expressions, constants, or register variables.
The unary operator * is the indirection or dereferencing operator;
when applied to a pointer, it accesses the object the pointer points to.
The declaration of a pointer variable is:

```
[datatype] *[variable name]
```
for example: `int *ip;`

means `ip` is pointer variable which reference an integer variable. i.e. `*ip` in an int, and `ip` is an pointer which stores an address value.
Initialization of a pointer

- There is no legal default value to a pointer variable. You have to initialize it before using it.
- C guarantees that zero is never a valid address for data, so a pointer of value of zero can be used to signal an abnormal event.
- The symbolic constant NULL is often used in place of zero which is defined in `<stdio.h>`.
- A pointer has to be initialized to the address of an existing variable before any meaningful using. For example:

```c
int i, *ip;
ip = &i;  // or int i, *ip = &i;
*ip = 3;
```

- This is illegal

```c
int *ip;
*ip = 3;
```
The *ip in above case is just an integer variable, so it can be put into the expression where integer can be put in. For example:

```c
*ip = * ip + 10;
*ip += 1;
*ip <<= 2;
*ip < 2;
++*ip;
(*ip)++; // means (*p) = (*p) + 1
*ip++; // means *(ip = ip + 1)
```

because unary operators like * and ++ associate right to left.

these are all legal expressions.
Since C passes arguments to functions by value, there is no direct way for the called function to alter a variable in the calling function.

With pointer it can.

Exercise:
Write a function that swap two integer variables’ value in the caller scope.
In C, there is a strong relationship between pointers and arrays.

In fact array variable is just one type of pointer. It can be directly assigned to a pointer variable. For example:

```c
int a[10] = {-1, -2}, *p = a;
printf("%d\n", *p);
```

Besides a is just storing the address of the first element of a.

```c
int a[10] = {-1}, *p = a;
printf("%d\n", a == &a[0]);
// what will be print out ?
```

And p can also be applied array subscripting like:

```c
printf("%d\n", p[1]); // or
printf("%d\n", *(p+1));
```
In evaluating `a[i]`, C actually converts it to `*(a+i)` immediately; the two forms are equivalent.

`&a[i]` and `a+i` are also identical.
There is one difference between an array name and a pointer that must be kept in mind.

A pointer is a variable, so \texttt{p=a} and \texttt{p++} are legal. But an array name is not a variable; constructions like \texttt{a=pa} and \texttt{a++} are illegal.

Array name is equivalent to a symbolic constant address value, and it has to be a stack address.

A pointer can reference to a heap address. We will see how later.
As formal parameters in a function definition, char s[] and char *s are equivalent.

It is preferred of the latter because it says more explicitly that the parameter is a pointer. That’s why you see a lot "char *s" in library function headers.

If one is sure that the elements exist, it is also possible to index backwards in an array; p[-1], p[-2], and so on are syntactically legal, and refer to the elements that immediately precede p[0].

Of course, it is illegal to refer to objects that are not within the array bounds.

Note: it is different in some other language which p[-1] means the last element of the array.
Address Arithmetic

- alloc
- free
String constant.

```c
char amessage[] = "now is the time"; /* an array */
char *pmessage = "now is the time"; /* a pointer */
```

- `amessage` is an array. Its individual characters within the array may be changed but `amessage` will always refer to the same storage.
- `pmessage` is a pointer, initialized to point to a string constant; the pointer may subsequently be modified to point elsewhere.
- All in all `amessage` is left value, while `pmessage` is a right value.

All in all `amessage` is left value, while `pmessage` is a right value.
char *lineptr[3];
lineptr[0] = "hello";

lineptr is an array of 3 elements, each element of which is a pointer to a char.
Two-dimensional Arrays

• Declaration and initialization.

\[
\text{[datatype]} \ [\text{name}][[m]][[n]];
\]
\[
\text{[datatype]} \ [\text{name}][[m]][[n]] = \{\{\}, \{\}\};
\]

For example:

\[
\text{static char daytab}[2][13] = \{
\{0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31\},
\{0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31\}
\};
\]
If a two-dimensional array is to be passed to a function, the parameter declaration in the function must include the number of columns; the number of rows is irrelevant:

```c
f(int daytab[2][13]) { ... }
```

It could also be:

```c
f(int daytab[][13]) { ... }
```

since the number of rows is irrelevant,

or it could be:

```c
f(int (*daytab)[13]) { ... }
```

The parentheses are necessary since brackets [] have higher precedence than *.

Without parentheses, the declaration:

```c
int *daytab[13]
```

is an array of 13 pointers to integers.